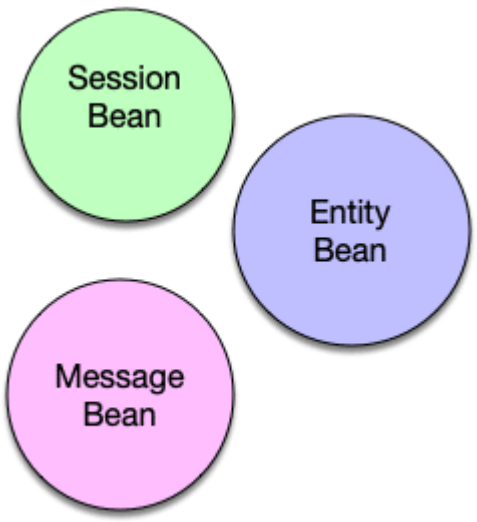
Tabla de Contenidos

- POJO , Java EE y su historia
  - Java EE y Tipos de Entidades
- POJO y Evolución
- Otros artículos relacionados

¿Que es un POJO? .Muchas personas me hacen esta pregunta cuando están comenzando a trabajar con el lenguaje Java ya que se encuentran en situaciones que gestionan la capa de Persistencia o capas de comunicación REST que hacen uso intensivo de POJOS o Plain Old Java Objects . ¿Para que sirve un POJO y de donde viene su nombre ? .Vamos a explicarlo .

POJO , Java EE y su historia

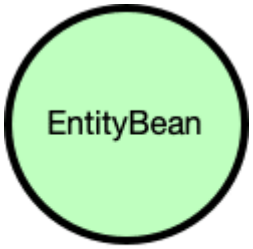
Hace ya muchos años cuando nos encontrábamos en las primeras versiones del lenguaje Java . Sun presentó su plataforma Java 2 EE . Esta plataforma servía para desarrollar aplicaciones Enterprise con servicios transversales . Las primeras versiones de la plataforma Java 2 EE fueron muy innovadoras en cuanto a tecnologías y desarrollaron 3 componentes fundamentales de la Arquitectura Enterprise .



Java EE y Tipos de Entidades

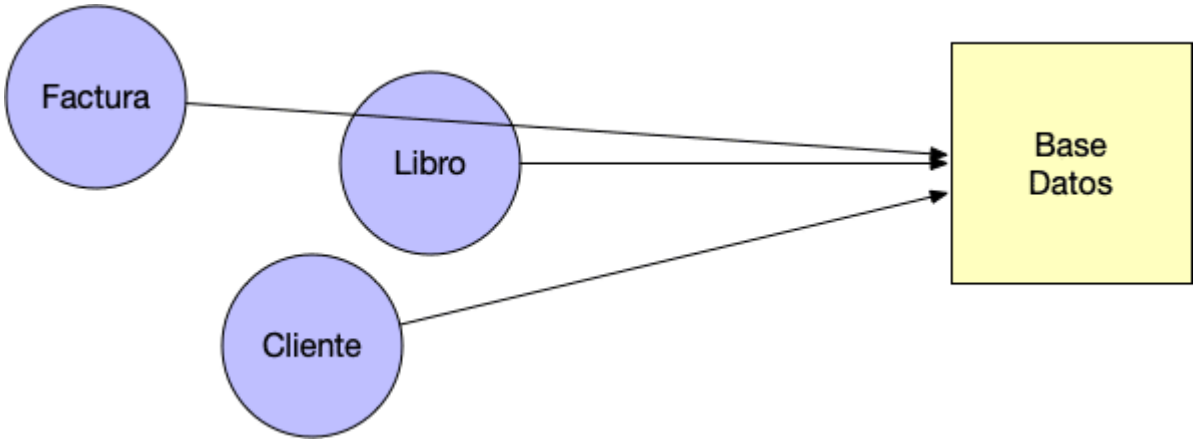
Los Session Beans , los MessageBeans y los Entity Beans . Cada uno de estos componentes tenia una funcionalidad muy concreta que desarrollar.

- Los Session Beans eran componentes que se usaban para gestionar tareas muy diversas desde ordenaciones listados , generación de ficheros etc , Eran componentes que se centraban en la funcionalidad que la aplicación debía implementar .
- Por otro lado teníamos los Message Beans que eran componentes orientados a Mensajería y colas asíncronas . Estos componentes se encargaban de realizar tareas asíncronas complejas que tardaban bastante tiempo en ser ejecutadas y había que encolar por ejemplo la generación de todas las facturas de la empresa en formato pdf.
- Por último existían los Entity Beans que eran objetos que se automatizaba su persistencia contra una base de datos . Estos objetos cuando fueron diseñados tenían una alta complejidad a la hora de realizar su persistencia en la base de datos y surgieron muchas voces en su contra ya que implementaban unos interfaces excesivos



POJO y Evolución

Para solventar este tipo de problemas nació Hibernate como framework en donde la persistencia de clases en la base de datos se hacía sobre clases Java normales y corrientes . Es decir Hibernate sin tener que hacer que la clase implemente interfaces adicionales es capaz de persistirla de forma transparente contra la base de datos.



Pronto esta forma de persistir clases normales comenzó a tener mucho éxito y se acuño el nombre de POJO (Plain Old Java Object) para aplicarlo con un enfoque de patrón de diseño de Persistencia. A partir de ese momento Hibernate fue ganando enteros como enfoque correcto de persistencia y los estándares se amoldaron a él . A partir de Java EE 5 se uso como capa de persistencia JPA o Java Persistence API que como especificación se baso en los conceptos aprendidos en Hibernate y en el manejo de POJOS. A partir de ese momento el concepto de amplio a otras plataformas como .NET en donde se uso el de POCO (Plain Old CLR Object). Por lo tanto un POJO es simple y llanamente una clase Java clásica que define un objeto de negocio :

```
package com.arquitecturajava;

import java.io.Serializable;

public class Persona  implements Serializable{

        /**
         *
         */
        private static final long serialVersionUID = 1L;
        private String nombre;
        private String apellidos;
        private int edad;
```

```
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.edad = edad;
	}
	public Persona() {
		super();
	}
}
```

Normalmente estas clases para encagar de forma natural con los frameworks de persistencia deben incluir un constructor por defecto e implementar Serializable.

Otros artículos relacionados

- ¿DTO vs Entity? en Domain Driven Design
- ¿JPA vs Hibernate?
- Un ejemplo de JPA Entity Graph
- WikiPedia
- StackOverFlow